Supporting Sustainability and Technical Debt-Driven Design Decisions in Software Architectures

***Daniel Guamán*[1,2]**
[1]*Universidad Politécnica de Madrid*
Madrid, Spain *da.guaman@alumnos.upm.es*
[2]*Universidad Técnica Particular de Loja*
Loja, Ecuador ***daguaman@utpl.edu.ec***

***Jennifer Pérez*[1]**
[1]*Universidad Politécnica de Madrid*
Madrid, Spain ***jenifer.perez@upm.es***

Abstract

Degraded software usually incurs higher energy consumption, therefore suboptimal decisions in software architectures may lead to higher technical debt and less sustainable software products. There are metrics and tools to calculate technical debt and energy consumption of software, but it is required to provide mechanisms to store their relationship and how they change depending on the design decisions. In addition, there are different models for calculating the same metric and different metrics to measure technical debt and power consumption, and software engineers require selecting the most suitable model and metric depending on the software product context. This work presents a metamodel called ARCMEL to provide the required base of knowledge for supporting green-aware design decisions and to flexibly configure and select metrics and their models. ARCMEL has been implemented as part of the ARCMEL SCAT tool. Its validation is also presented in terms of completeness and flexibility.

Keywords: Green Metrics, Technical Debt, Software Architecture, Design Decisions, Metamodel.

1. Introduction

Everyday sustainability increases its criticism in our society, which has led companies to be aware on their social, economic, environmental, technical and individual dimensions [3]. The technical dimension addresses Green IT [19] by improving the energy efficiency of both, software and hardware. Traditionally, energy efficiency research has focused on reducing power consumption at the hardware level. But currently, the Green IT and software sustainability development have become fundamental to encourage an efficient use of technological resources in order to reduce environmental impact [14]. Green IT and Green Software [19] include practices and criteria in the design, development, implementation, and use of the software, which allow to identify and evaluate metrics associated with sustainability, reduction in the consumption of technological resources, and energy efficiency of software [7]. In fact, the technical dimension of the sustainability manifesto [3] refers to the longevity of information, systems, and infrastructure and their adequate evolution with changing surrounding conditions. It includes maintenance, innovation, obsolescence, data integrity, etc.

Venters et al. [41] and Villa et al. [43] state that maintainability is a quality attribute that influences software sustainability. In particular, Technical debt (TD) is a critical factor in software maintainability [7], because it is a concept that studies the economic consequences due to the increase or decrease in the quality of the software generated by design and code decisions to prioritize certain aspects of the business [39]. Maintenance changes may generate technical debt [22], considered as the debt that originates from problems or anomalies in the code. Several studies have highlighted the negative effects of uncontrolled technical debt on software development [39]. As a result, it is necessary to provide mechanisms, methods and tools to build sustainable software products that improve technical debt and energy efficiency by improving quality and reducing

complexity [19].

At the code level, these TD and sustainability problems are classified as code smells [15] and energy smells [42] that also increase the software energy consumption. The term code smells are manifestations of poor implementation and design flaws that can degrade code and difficult the software maintainability [9],[11]. Energy Smell is an implementation that appears at the source code level (code patterns), design, or architectural level that makes a sub-optimal usage of the hardware resources. Consequently, it provokes an energy debt [11], which is considered as the amount of unnecessary energy that a software system uses over time, due to maintaining energy code smells for periods. There are different models for calculating these metrics and different metrics to measure technical debt and power consumption, and software engineers require mechanisms for selecting the most suitable model and metric depending on the software product context. A good practice to reduce code smells, energy smells and the energy consumption is to use models that calculate energy consumption from technical debt implemented by tools, and techniques that allow findings errors and evaluate code metrics to carry out refactoring and maintainability activities of the software to improve code and energy smells. However, currently, to use only one approach and model to evaluate the software and build decisions in terms of the relationship between technical debt and sustainability is not feasible [40]. This problem is due to the existing models and the tools that support them do not store all the required information to support these TD green decisions or are specific to the domain or application context. Therefore, it is necessary a domain-independent model to manage the knowledge of software architectures, design decisions, and the TD and green metrics in order to estimate TD and energy consumption and support the design decision-making process. In this work, we present the metamodel ARCMEL to support these needs. This solution has been constructed using the Model-Based Engineering approach (MBE) [4], since models automate development tasks and stimulate learning and reasoning capabilities, which are essential for decision-making support. Specifically, this work presents the abstract syntax that defines the ARCMEL metamodel, as well as its concrete syntax based on a web graphic language implemented as part of the automated static analysis tool SCAT (Source Code Analysis Tool) of ARCMEL[17], called ARCMEL SCAT. In addition, this work illustrates how the concrete syntax of the metamodel allows to carry out efficiently the load, extraction, and analysis of software applications in technical debt and energy consumption contexts. Finally, the paper presents an evaluation of the metamodel and its tool support in terms of (i) completeness to provide the knowledge derived from architecture, design decisions, and code and energy smells, and (ii) flexibility to customize the calculation models of metrics.

The paper is organized as follows: the works related to modelling, technical debt, and energy consumptions are presented in Section 2. The ARCMEL metamodel is described in Section 3. Section 4 presents the concrete syntax through the tool ARCMEL SCAT. Section 5 details the experimental validation that has been performed to validate the metamodel. Section 6 presents the threats to validity, and finally, the conclusions and future work are presented in Section 7.

2. Related Work

Model Based Engineering (MBE) [6] uses models raising the abstraction level of applications to apply generation and automation techniques for the analysis and development of software systems. This higher level of abstraction is supported by Meta-Object Facility (MOF) [26] through the definition and manipulation of metamodels. These models help us to understand complex problems and their potential solutions and stimulate learning and reasoning. Understanding the architectural decisions made during the design of an architectural solution is critical. As a result, the software architecture community has worked in providing metamodels, formal representations and tools for supporting the design rationale; i.e. the Architectural Knowledge (AK) [37].

AK aims to maintain knowledge and documentation of the fundamentals, assumptions, and other factors that together determine the significant design and implementation decisions of software [36]. The standard ISO/IEC/IEEE 42010 [35] defines the reference

AK model [8], which provides a model to characterize the constructors for documenting, describing and representing software architectural decisions. This metamodel mainly focuses on architectural elements and design decisions as model constructors. There are other AK models that also store knowledge extracted from the code. From these models, the metamodel of Stevanetic et al. [34] takes a step forward introducing tactics to address design decisions driven by quality attributes. In addition, the work of Venters et al. [40, 41], states that the quality attributes influence in software architecture sustainability, and therefore, it is necessary to have metamodels that address both sustainability and other quality attributes. In addition, Villa et al. [43] established maintainability as one of the quality attributes that influence software architecture sustainability. Therefore, since technical debt is a critical factor in software maintainability, metamodels to support the architectural knowledge of technical debt and sustainability are required.

Regarding sustainability, the work of Carrillo et. al [9] takes a step forward in the area by presenting a model where green design decisions are considered, however although that they emphasize the relevance of technical debt, it is provided by external components without being modelled as part of their metamodel and the relationships between them. The positive and negative impacts of sustainability in the software development life cycle can be determined using a model proposed by [1] wherein the design, as one of the phases of the model, suggests using design principles such as (modularization, abstraction, cohesion, coupling) because energy consumption is reduced. At the development level, in the model it is recommended to take into account data structures and algorithms, avoid duplicate code and uncontrolled data flow derived from the unnecessary use of control statements, thereby improving the maintainability and reducing complexity, technical debt, and energy consumption.

Regarding Technical Debt, the models work of Li et al. [25] recover technical debt information applying Knowledge Extraction from Source Code (KNESC) [2], an approach that aims to extract and analyse the knowledge embedded in the source code through automated reverse engineering processes. However, these models do not address sustainability.

Quality-aware AK Metamodels specify quality metrics as part of them and play an important role since they help to estimate and evaluate software quality in a quantitative way. Regarding green quality metrics, Welter et al. [44] expose that green metrics help to monitor and evaluate software in an ecological context, where the evaluation depends on the structure of the application and the IT infrastructure. As result, there are already works that affirm that there are code metrics, whose value impacts energy consumption [3], [10], [13], [28, 29], [24]. In fact, although software power consumption refers to the runtime execution of software applications and requires a dynamic analysis, there already are green metrics to estimate the energy consumption of a software application without the need of being executed. The metrics Executed Instruction Count Measure (EIC), and Memory Access Count Measure (MAC) use techniques and automated static analysis (ASA) tools to compute and quantify values that allow estimating the Software Energy Metric (SEM) [10]. In the same static analysis context, the metric Energy Wasting Rate [29] is used to identify the classes, attributes, methods, or interfaces that in a software application should be refactored to reduce the energy consumption. Although these metrics require the computation of ASA tools, they also need to run the application for a while to obtain the corresponding energy consumption values using tools for measuring power consumption. Therefore, the relationships that these metrics define between the code and energy consumption are the knowledge base for sustainability decisions. In addition, the SOLID Object-Oriented Design Principles help reduce energy consumption, and a set of metrics that help analyse the pillars of OOP (abstraction, encapsulation, inheritance, and polymorphism) [24].

Metrics can be simple or complex. Simple metrics can be calculated using the compiler or open-source libraries such as the number of packages, classes, lines of code, etc. Complex metrics measure characteristics at a higher level of abstraction through equations calculation models based on a set of simple metrics or other complex metrics. The metrics that evaluate quality attributes and estimate energy consumption are usually compound-

complex metrics that can be calculated by applying different equations models, for example, Cognitive Weighted Method Hiding Factor Complexity (CWMHF)[38], Cognitive weighted attribute hiding factor complexity metric [38], Catch area average Exception in a class [32], among others. However, currently, there are no tools that allow the customization of the calculation model depending on the context and needs. The current TD measurement tools as SonarQube, and energy consumption estimation tools such as JouleMeter [21], RAPL [12], and PowerAPI [5], support the extraction or calculation of metrics and their values storage, but they are not able to address both TD and green [18], and they do not allow customization of calculation models.

From this related work analysis, this work takes a step forward by presenting the ARCMEL metamodel to support the green and TD design decision-making. ARCMEL defines a set of model packages that properly related allow managing the architectural knowledge and metrics of both technical debt and estimation energy consumption. In addition, this metamodel allows the customization, management, and storage of calculation equations configured by simple and complex metrics according to the context and needs of each software engineer.

3. ARCMEL Metamodel: Abstract syntax

This section presents ARCMEL as a metamodel to specify not only the design decisions of software architectures but also the metrics that support them to be technical debt and sustainable-aware. This metamodel is also characterized by providing flexibility to customize the equation models applied by each metric and to store them. This metamodel is composed of four packages: *ArchitectureModel, ClassModel, MetricModel, and SmellModel* (see Fig. 1). The packages *ArchitectureModel* and *ClassModel* define the software system at different levels of abstraction, specifically from source code to software architecture, and their design and rationale. On the other hand, the *MetricModel* and the *SmellModel* are related to quality attributes measurement, specially they allow to store knowledge among the relationships between code smells and green metrics. Each package is described in detail in the following subsections.

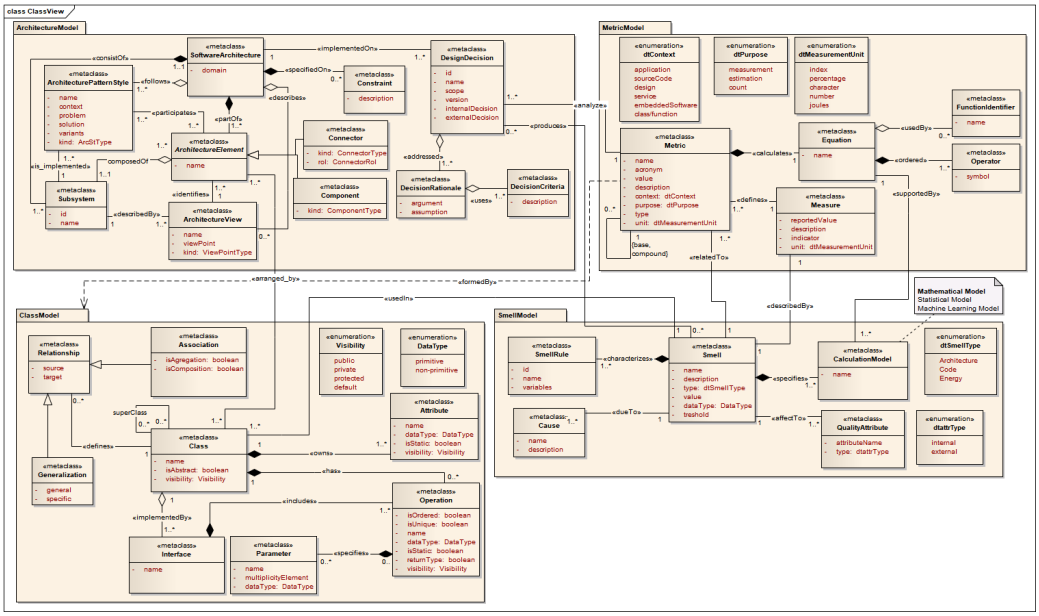


Fig. 1. ARCMEL Metamodel

3.1. Package Architecture Model

The package *ArchitectureModel* is based on the standard ISO/IEC/IEEE 42010 to define the software architecture and its design decisions to store and analyse architecture knowledge. *ArchitectureView*s commonly organize the software architecture design into the different points of view that represent the software system (see Fig. 2). In addition, software architectures are structurally implemented by using architectural styles, architectural patterns, and design patterns, that can be applied in a unified or combined way. These patterns and styles are defined with the *ArchitecturePatternStyle* metaclass (see

Fig. 2). In addition, the *Constraints* that architectures must fulfil have to be taken into account (see Fig. 2). Regarding *ArchitectureElement*s, they are the fundamental pieces to build software systems and can be *Component*s and *Connector*s depending on their role of data processing or connection orchestration, respectively. At the same time, these architectural elements can be composed into more complex *Subsystem*s (see Fig. 2). Finally, the architectural knowledge is described in terms of *DesignDecision*s, which are complemented with the associated rationale that made to take the decision (*DecisionRational* metaclass) and their corresponding *DecisionCriteria* (see Fig. 2).

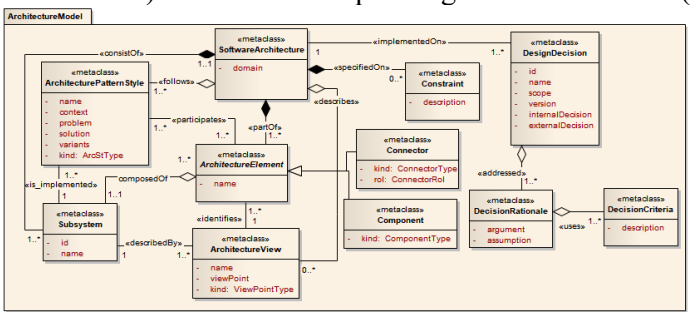


Fig. 2. Package: Architecture Model

3.2. Package Class Model

The *ClassModel* package has been defined to store source code characteristics (reserved words, lexical rules, syntactic rules, and tokens) of software architectures (see the relationship between the metaclasses *Class* and ***ArchitecturelElement*** Fig. 1) to be used as input for the configuration, parameterization, and calculation of simple and complex metrics used to analyse technical debt and estimate energy consumption. This package has been defined using the class diagram metamodel presented by Paige et al. [27] as a reference model. This package describes all metadata of a *Class* and its *Interface*s and *Relationship*s, i.e. *Generalizations* and *Association*s to describe their different properties and relationship semantics (see Fig. 3). In addition, ARCMEL stores the Attributes and Operations of classes as well as their corresponding Parameters, Visibility, and Datatypes, which are required to calculate the values of some metrics types (see Fig. 3).

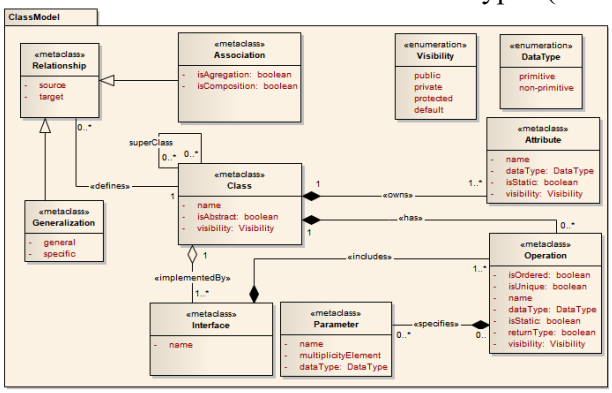


Fig. 3. Package: Class Model

3.3. Package Smell Model

The consequences of TD are known as smells, which have a negative effect on energy consumption or quality attributes that affects the current or future functionality of the software, and must be corrected during the software maintenance or evolution. The *Smell Model* Package is in charge of modelling this knowledge, the *Smell* and its *Causes* (see Fig. 4). But also, ARCMEL requires to know what generates technical debt, i.e design decisions, actions (or lack of actions), or events that trigger the existence of that element of debt, to learn and generate new knowledge about TD that may improve the architectural design-decision process. This information is recovered thanks to the relationships between the metaclasses *Smell* and *DesignDecision,* and *Smell* and *Class* (see Fig. 1). The *SmellModel* Package can store the *SmellRule*s of a *Smell,* which defines the bug, vulnerability, and severity properties (see Fig. 4). The *Smells* affect negatively the *QualityAttribute*s, especially when existing code smells, design smells, or energy smells

(*dtSmellTypes*). This negative influence can be minimized if our model can provide this information and relationship after these smells were identified and stored in our metamodel using reverse engineering processes. ARCMEL is also aware of the existing variability of TD and Energy *Smells CalculationModel*s. As a result, the model also stores the equations used for calculating and *Smell* with the *CalculationModel* metaclass (see Fig. 4).

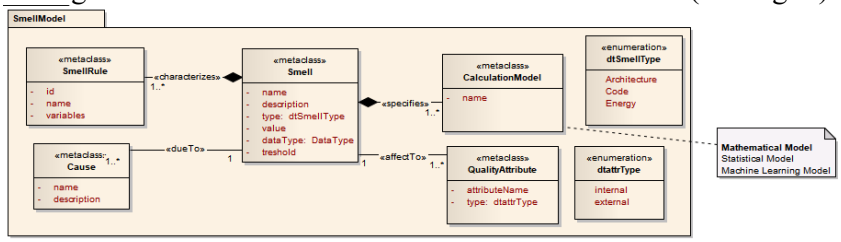


Fig. 4. Package: Smell Model

3.4. Package Metric Model

The *MetricModel* package defines the *Equation*s, *Operator*s, and *FunctionIdentifier*s that allow obtaining quantitative values of *Metric*s to estimate the energy consumption and architectural sustainability from a TD perspective (see Fig. 5). This relationship is achieved thanks to the relationship between the *CalculationModel metaclass* and the *Equation* Metaclass and the *Metric* metaclass and *ClassModel* (see Fig. 1). However, despite the fact this *MetricModel* has been conceived for a sustainability purpose, its structure is generic enough to be used for any kind of *Metric.* In addition, the *MetricModel* thanks to the decomposition of complex *Metrics* into simple *Metrics* (see the reflexive relationship, Fig. 5) and the simple *Metrics* into *Equation*s, *Operator*s, and *FunctionIdentifier*s provides the required flexibility to configure the *CalculationModel*s storing the applied calculation *Equation* for each *Metric.* Thanks to this detailed information decomposition, the *MetricModel* can answer questions such as what metric is defined for evaluating a specific architectonical design decision context? What are the simple metrics that constitute a complex metric? or What is the measurement and unit of measurement that is obtained as a result? This last question is feasible thanks to the information stored by the *Measure* metaclass and the attributes of the *Metric* metaclass: *name*, *description*, *unit* of *measurement*, and metric *type*. In addition, it is required to define the context and purpose where the metric will be applied, which is recovered by the defined enumerations (see Fig. 5).

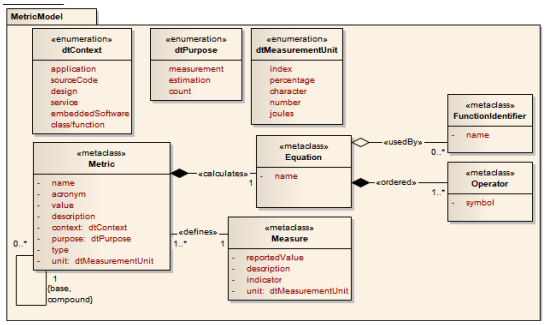


Fig. 5. Package: Metric Model

4. ARCMEL SCAT (Source Code Analysis Tool): Concrete syntax

The smells affect negatively the *QualityAttribute*s, especially when existing code smells, design smells, or energy smells. This negative influence can be minimized when using reverse engineering processes and tools first we identify smells and then apply good practices and specific refactoring techniques to solve them. The ARCMEL SCAT allows both: to apply the reverse engineering process to identify the information and smells [17], and to store them in the ARCMEL metamodel to support the rational and learning process that and TD and green aware architectural design decision process requires.

ARCMEL SCAT implements the concrete syntax of the ARCMEL metamodel as a web domain-specific language to avoid the learning curve and providing a friendly tool that could be easily integrated in the software engineer's framework following the guidance of

Capilla et al. [8], who suggest that any custom tool should be lightweight and descriptive instead of prescriptive, otherwise it will never be adopted by software engineers.

To extract and calculate the value of the metrics in a flexible and customizable way (see Fig. 6). ARCMEL SCAT, through the web concrete syntax language, provides the metric customization at two levels of abstraction: value and definition. The **value customization** allows the software engineer to establish the positive and negatives values of a metric depending on the product and the requirements for both TD and Green (see Fig. 7.a). This flexibility is required because the same value may be bad or good depending on the software product and also can be different for measuring TD or Green. The **definition customization** allows the software engineer to configure the characteristics, measurements, metrics, and equations according to his her needs, or even, to creating new ones (see Fig. 7.a and Configuration.Fig. 6). All this information is stored in the *MetricModel* of ARCMEL. Regarding the SCAT part of ARCMEL SCAT, it implements an automated static analysis that extracts the knowledge from the source code and validates the grammar of the application under analysis. After this validation, the information is automatically stored in the *ClassModel* of the ARCMEL metamodel (see Fig. 6).

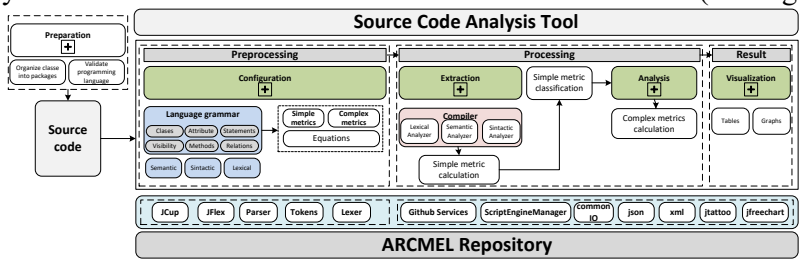
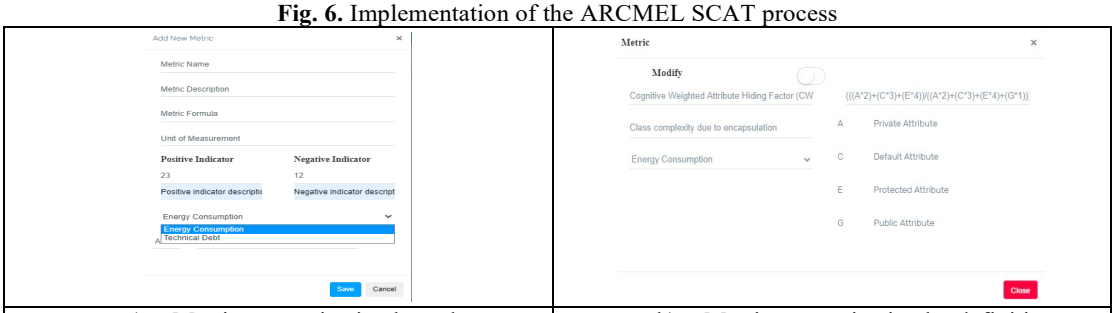


Fig. 6. Implementation of the ARCMEL SCAT process



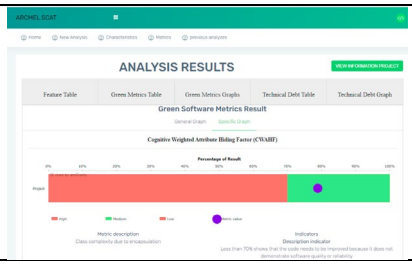
a) Metric customization by value

b) Metric customization by definition

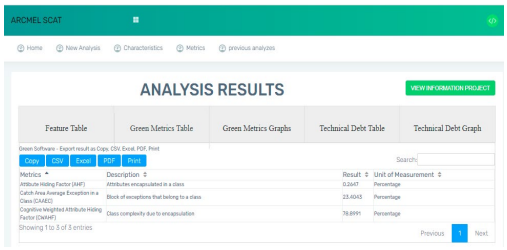
Fig. 7. ARCMEL SCAT configuration of metrics

The extraction automated process uses the settings made through the configuration to calculate simple metrics. Most of these metrics are calculated using basic and complex arithmetic operators and operations, that allow the count and sum of characteristics or lexical rules found in the code, and that are necessary to configure and calculate the values associated with these metrics. The *Measures* are stored in the *MetricModel* (see Extraction, Fig. 6). At this point, the analysis calculates the complex *Metrics* and store their values as *Measures* in the *MetricModel* (see Analysis, Fig. 6). Finally, *Measures* resulting from the extraction and analysis of simple and complex metrics are visualized by SCAT ARCMEL using formats such as tables and/or graphs to support the software engineer interpretation and decision-making from a technical debt and energy consumption analysis (see Visualization, Fig. 6 and Fig. 8). Additionally, to this visualization the software engineering introduces the architectonical information that cannot be automatically extracted from the code to address more complex and architecture design decisions.

As a result, ARCMEL SCAT extends the capabilities of other ASA tools such as SonarQube [31], JDepend [20], or Structure Analysis for Java (STAN) [33] that can customize the positive and negative values of a metric for TD. On the one hand, ARCMEL SCAT provides this value customization capability not only for TD, but also for Green. On the other hand, it adds the metric definition customization by the configuration of code metrics and the definition of new ones. As a result, ARCMEL SCAT is able to introduce metrics for providing the required information for dynamic estimation models of power consumption of software application such as [29], [32], [38].



a) Green Graph Visualization



b) Green Table Visualization

Fig. 8. ARCMEL SCAT Analysis Visualization

5. Validation

To validate the ARCMEL metamodel and its ARCMEL SCAT tool, we have conducted a study to determine the completeness of the knowledge provided from the architecture, design decisions and code and energy smells, and the flexibility to customize the calculation models of metrics.

5.1. Research objectives and questions

To address the validation of completeness and flexibility of ARCMEL, we have defined the following research questions:

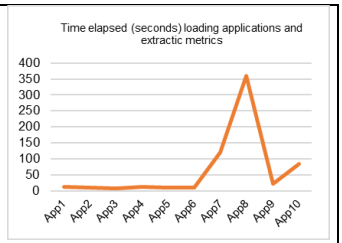
- **RQ1:** Is ARCMEL SCAT able to extract and store the required architectural information, characteristics, and metrics to support design decisions and estimation based on technical debt and energy consumption?
- **RQ2:** What is the flexibility degree of ARCMEL SCAT to configure and customize the characteristics and metrics for technical debt and energy consumption estimation?

5.2. Data Collection

To conduct the validation, we used the most extended ASA tools SonarQube, JDepend and STAN, and a set of applications. These applications are required to demonstrate that ARCMEL SCAT can extract characteristics and metrics from source code, to store and analyse them thanks to the ARCMEL metamodel (see Fig. 9.b). To that end, the applications were searched using inclusion criteria of the Java programming language and the use of different architectural patterns or styles to avoid architectural bias. The search was carried out in a public repository where the software code is accessible and written by different programmers to avoid programming bias. Applying these inclusion criteria, 10 applications were selected and downloaded from GitHub (https://github.com/) for this study. Each downloaded application was analysed manually and compiled using NetBeans IDE 8.0. From this analysis, we obtained the characteristics of the applications presented in (see Fig. 9.a).

Application location	Applications characteristics						
URL Git	App Name	IdApp	Type	Repository	LOC	Size	Architecture
https://github.com/{blind}/ExpressRace.git	ExpressRace	App1	Desktop	MySql	3500	S	Layers
https://github.com/{blind}/puremvc.git	PureMVC	App2	Mobile	None	2000	S	MVC
https://github.com/{blind}/rest.git	Rest	App3	Web	MySql	1697	S	MVC
https://github.com/{blind}/SigPC.git	SigPC	App4	Desktop	MySql	3511	S	Layers
https://github.com/{blind}/SaleTickets.git	SaleTickets	App5	Desktop	MySql	2739	S	Layers
https://github.com/{blind}/WebServiceJava.git	WebService	App6	Web	MySql	2737	S	MVC
https://github.com/{blind}/ClinicProject.git	ClinicProject	App7	Web	MySql	3800	S	Layers
https://github.com/{blind}/java-client-library.git	Java Client Library for DeployR	App8	Desktop	None	8100	S	Publisher/suscriber
https://github.com/{blind}/java-patterns.git	Java Patterns	App9	Desktop	None	609	XS	Layers
https://github.com/{blind}/greenhouse.git	Greenhouse Application	App10	Web	MySql	6800	S	MVC

a) Characteristics



b) Extracting time

Fig. 9. Applications under study

5.3. Results

RQ1: Is ARCMEL SCAT able to extract and store the required architectural information, characteristics, and metrics to support design decisions and estimation based on technical debt and energy consumption?

RQ1 aims to analyse the completeness degree of the information that the metamodel is able to store and manage in order to support the estimation of TD and energy consumption. To that end, we have compared ARCMEL SCAT with SonarQube, JDepend and STAN to determine the number of metrics that are able to currently manage. From the detailed information of Table 1 and the synthesis of the results (see Fig. 10), it is possible to conclude that ARCMEL SCAT is currently the most complete tool providing information about metrics that impact on TD and

green to take green and TD design decisions and support the estimation of energy consumption. In addition, it is important to understand that this analysis completeness is an example of the current information managed by the metamodel, but this can be extended thanks to the flexibility of the model of adding and creating new metrics and estimation models. This flexibility is feasible thanks to the completeness of the information recovered by the metamodel. In order to illustrate the completeness of the metric information, we are going to use as example the Cognitive Weighted Attribute Hiding Factor (CWAHF) metric [38], which is an extension of the Attribute Hidding Factory (AHF) metric [38] that it is used to estimate energy consumption (see Table 2). In [38], this metric is manually calculated from source code characteristics to evaluate the architectural level complexity caused by the encapsulation of the attributes to calculate the cognitive complexity. ARCMEL SCAT can automatically compute the CWAHF metric. To calculate the metric, ARCMEL SCAT requires their configuration and then extracts the required information from the application. In this case, this data are the class information, its attributes and their corresponding type, which are instances of the metaclasses *Class* and *Attribute* from the *ClassModel* package (see Fig. 3). The CWAHF metric is configured using the package *MetricModel* since it requires simple metrics as the number of classes, the number of attributes, and the number of attributes of each type required by the formulae (see Table 2). In addition, the metric is associated with a *dtContext*, *dtPurpose* and *dtMeasurementUnit*, which are *SourceCode*, *Estimation* and *Percentage*, respectively. This metric is defined in ARCMEL SCAT as an *Energy dtSmellType* and it is associated with *Maintainability* as the effect of *Smell* and structural *Complexity* as *QualityAttribute* [41], [43].

Table 1. Metrics supported by ASA Tools: SonarQube, STAN, JDepend and ARCMEL SCAT

	Count/Size				Duplications				Complexity		Robert C. Martin							CK-Metrics					
Metric	Lines of Code	Functions/ Methods	Classes	Statement	Density	Duplicated files	Duplicated lines	Duplicated blocks	Cyclomatic Complexity	Cognitive Complexity	Efferent Coupling (Ce)	Afferent Coupling (Ca)	Instability (I)	Abstractness (A)	Distance from Main Sequence (D)	Concrete Classes Count	Abstract Class Count	Weighted Method per Class (WMC)	Depth of Inheritance Tree (DIT)	Num Of Children (NOC)	Coupling Between Objects	Response for a class (RFC)	Lack of Cohesion in Methods (LCOM)
Tool	□△☆			□△	□☆		□		□△☆	□				△☆○					△				△☆
Characteristic/metric that impact on TD or is used to estimateTD	[42], [10]	[42] , [10]	[42], [10], [29]	[42], [52]	[42], [10]				[42], [28]	[42]	[2]	[2]	[2]	[2]	[2]	[2]	[2]	[41]	[41]	[41]	[41]	[41]	[41]
Characteristic/metric that impact on energy consumption	[28]	[28],[31], [10], [42]	[29], [42]						[28]		[28], [2]	[28], [2]	[28], [2]	[28], [2]				[28]			[28]	[28]	[28]

	Method Metrics						Attribute Metrics				Quality Metrics							
Metric	Legacy methods	Default methods	Private methods	Protected methods	Public methods	Overrriden methods	Default attributes	Private attributes	Protected attributes	Public attributes	Catch Area Average Exception in a Class (CAAEC)	Cognitive Weighted Attribute Hiding Factor (CWAHF)	Cognitive Weighted Method Hiding Factor Complexity Metric (CWMHFC)	Cognitive Weighted Polymorphism Factor (CWPF)	Method Hiding Factor (MHF)	Attibute Hiding Factor (AHF)	Tight Class Cohesion (TCC)	Weight Method Count
Tool	☆						☆						☆					
Characteristic/metric that impact on TD or is used to estimateTD			[41]		[41]			[41]		[41]			[34]	[34]	[41]	[41]	[41]	[41]
Characteristic/metric that impact on energy consumption	[34]	[34]		[34]		[34], [28]	[34]		[34]		[35]	[34]				[28]		[28]

Legend: □ --- SonarQube; △ --- STAN; ○ --- JDepend; ☆ --- ARCMEL SCAT

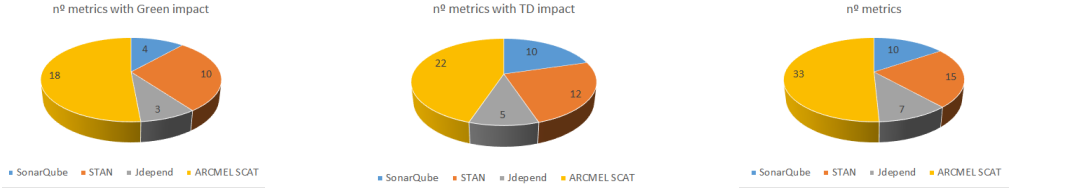


Fig. 10. Metrics support by ASA Tools

Table 2. Cognitive Weighted Attribute Hiding Factor (CWAHF) metric

$$CWAHF = \frac{\sum_{i=1}^{TC} A_n(C_i)}{\sum_{i=1}^{TC} A_n(C_i) + \sum_{i=1}^{TC} A_v(C_i)} \quad (1)$$

$$\sum_{i=1}^{TC} A_n(C_i) = \sum_{i=1}^{TC} A_p(C_i) * CW_{pa} + A_d(C_i) * CW_{da} + A_t(C_i) * CW_{ta}$$

$$\sum_{i=1}^{TC} A_v(C_i) = \sum_{i=1}^{TC} A_u(C_i) * CW_{ua}$$

$A_p(Ci)$ = Number of private attributes in the class
$A_d(C_i)$ = Number of default attributes in the class
$A_t(C_i)$ = Number of protected attributes in the class
$A_u(C_i)$ = Number of public attributes in the class
CW_{pa} = Private attribute cognitive weight
CW_{da} = Default attribute cognitive weight
CW_{ta} = Protected attribute cognitive weight
CW_{ua} = Public attribute cognitive weight
TC = Number of classes

An added value of ARCMEL is the ability to configure indicators values associated with the metrics (see Fig. 7.a). For example, the CWAHF metric has been configured as the cognitive weight for public attributes 1, private attributes 2, default attributes 3 and for protected attributes 4. The results of this value configuration is illustrated in Fig. 8.a. Finally, we have automatically extracted the information of the 10 selected applications to determine if ARCMEL SCAT can extract, store, analyse and visualize them without problems (see Fig. 9.b). This process also has been reproduced in SonarQube, JDepend and STAN and it was successfully executed in all of them (see Table 1).

RQ2: What is the flexibility degree of ARCMEL SCAT to configure and customize the characteristics and metrics for technical debt and energy consumption estimation?

The flexibility of ARCMEL is evidenced through the functionality of ARCMEL SCAT that allows the configuration of metrics defined by equations a fine granularity level (see Fig. 7.b).

The metric configuration allows the software engineer to customize equations using mathematical or logical operators that combine operands as code characteristics, simple metrics, or complex metrics (see the metaclasses *Equation*, *FunctionIdentifier*, *Operator*, *Measure* and *Metric* Fig. 5). Is important to mention that these customized equations and metrics can be associated with a type of smell (code, energy) (see Fig. 4) to take design decisions in terms of technical debt or energy consumption (see Fig. 8).

This flexibility has allowed ARCMEL to support a higher number of complex metrics than SonarQube, JDepend and STAN such as CWAHF, CAAEC, CWPF, etc. (see Table 1 and Fig. 10). To illustrate this flexibility, we have configured 6 complex metrics for the 10 applications under study and we have defined their positive and negatives values to evaluate technical debt and green consumption. The results of Table 3 reveal that ARCMEL SCAT was able to obtain all the results. The table shows different values that depends on the values and simple metrics used by the equations. For example, the positive and negative values of the metrics CWAHF and CWPF were configured in a range [0..100], where the values from 0 to 50 are considered negative and suggest applying refactoring to improve energy consumption. On the other hand, if the values obtained are upper to 50 are considered as a positive indicator for technical debt and energy consumption. In the case of CAAEC the values have been configured in a range [0..50], and in the cases of Instability, Abstractness and Distance have been configured in a range [0..1].

Table 3. ARCMEL SCAT flexibility adoption to the 10 applications

	Applications									
Id application	**App1**	**App2**	**App3**	**App4**	**App5**	**App6**	**App7**	**App8**	**App9**	**App10**
Architecture of application	**Layers**	**MVC**	**MVC**	**Layers**	**Layers**	**MVC**	**Layers**	**Publisher/Suscriber**	**Layers**	**MVC**
Metrics	ExpressRace	PureMVC	Rest	SigPC	SaleTickets	WebService	ClinicProject	Java Client Library for DeployR	Java Patterns	Greenhouse Application
Cognitive Weighted Attribute Hiding Factor (CWAHF)	100	92	90	87	100	100	100	78.89	12	8
Catch Area Average Exception in a Class (CAAEC)	1.52	6.89	0.53	0	8.7	6.3	0	23.45	4.5	27.8
Cognitive Weighted Polymorphism Factor (CWPF)	13.63	6	0	0	13	20.84	12.67	23.94	63,4783	31.01
Instability (I)	0.25	0.8	0.45	0	0.36	0.46	0.65	0.13	0.17	0.45
Abstractness (A)	0.1	0.3	0	0.1	0.52	0.13	0	0	0.23	0.18
Distance From Main Sequence (D)	0.29	0	0.75	1	0.19	0.2	0.56	0.67	0	0.34

6. Threats to validity

To improve the **Internal validity** of the presented results, the selection of applications was based on the evidence described in the code or documentation from the GitHub repository. On the other hand, to collect the metrics, we used a local computer with OS Windows 10 to configure ARCMEL SCAT, SonarQube, STAN, Jdepend, and MySQL to avoid human intervention. To collect and extract the metrics through the tools, each application was downloaded from the GitHub repository, built, and executed. As a result, the automatic information management has avoided personal bias. **Construct validity** was ensured by following a systematic process in the conduction of the experimental study, and **external validity** was addressed by using 10 different software applications of Java with a wide variety of features. However, to improve the generality, it is necessary to increase applications including applications with M, L, and XL SIZES, as well as more variety of architectural styles and patterns, and other characteristics.

7. Conclusions

In this work, the metamodel ARCMEL is presented as a solution to represent and provide the required knowledge about software applications and their architectures related to green and technical debt metrics and smells to support green and TD design-decisions. This metamodel is supported by a concrete syntax through the ARCMEL SCAT tool, a static analysis tool that thanks to the fine level of knowledge representation provides high flexibility to configure equations formed by simple and complex metrics to obtain values customized for the context and requirements of each application under analysis. The completeness and flexibility of ARCMEL have been evidenced through the conduction of an experimental study of 10 applications extracting efficiently their required information. In future work, we plan to extend the experimental results by checking other kinds of applications and analysing the support of the tool during the design decision-making process. In addition, we are going to analyse the correlation between quality, complexity, and energy consumption from the metrics stored in ARCMEL in order to construct green estimation models from the characteristics of applications without the need of being executed.

Acknowledgements

This work is partially supported by Universidad Técnica Particular de Loja (Computer Science Department) and the Spanish Ministry of Economy and Competitiveness (MINECO) through the project CROWDSAVING (TIN2016-79726-C2-1-R).

References

1. Abdullah, R., Abdullah, S., & Tee, M (2014). Web-based knowledge management model for managing and sharing green knowledge of software development in community of practice. 8th Malaysian Software Engineering Conf., MySEC, pp. 210–215, 2014.
2. Azanzi, F. J., & Camara, G. (2017, October). Knowledge extraction from source code based on Hidden Markov Model: application to EPICAM. In *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)* (pp. 1478-1485). IEEE.
3. Becker, C., Chitchyan, R., Duboc, L., Easterbrook, S., Penzenstadler, B., Seyff, N., & Venters, C. C. (2015, May). Sustainability design and software: The karlskrona manifesto. Proceedings of the 37th Int. Conf. on Software Engineering-Vol. 2, 2015, pp. 467–476.
4. Beydeda, S., & Book, M. (2005). *Model-driven software development* (Vol. 15). V. Gruhn (Ed.). Heidelberg: Springer.
5. Bourdon, A., Noureddine, A., Rouvoy, R., & Seinturier, L. (2013). Powerapi: A software library to monitor the energy consumed at the process-level. *ERCIM News*, *2013*(92).
6. Brambilla, M., Cabot, J., & Wimmer, M. (2017). Model-driven software engineering in practice. *Synthesis lectures on software engineering*, *3*(1), 1-207.
7. Calero, C., & Piattini, M. (2017). Puzzling out software sustainability. *Sustainable Computing: Informatics and Systems*, *16*, 117-124.
8. Capilla, R., Jansen, A., Tang, A., Avgeriou, P., & Babar, M. A. (2016). 10 years of software architecture knowledge management: Practice and future. *Journal of Systems and Software*, *116*, 191-205.
9. Carrillo, C., Capilla, R., Zimmermann, O., & Zdun, U. (2015, September). Guidelines and metrics for configurable and sustainable architectural knowledge modelling. *ACM International Conference Proceeding Series, vol. 07-11-Sept, 2015.*
10. Chatzigeorgiou, A., & Stephanides, G. (2002). Energy metric for software systems. *Software Quality Journal*, *10*(4), 355-371.
11. Couto, M., Maia, D., Saraiva, J., & Pereira, R. (2020, June). On energy debt: managing consumption on evolving software. In *Proceedings of the 3rd International Conference on Technical Debt* (pp. 62-66).
12. David, H., Gorbatov, E., Hanebutte, U. R., Khanna, R., & Le, C. (2010, August). RAPL: Memory power estimation and capping. In *2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)* (pp. 189-194). IEEE.
13. Ergasheva, S., Khomyakov, I., Kruglov, A., & Succil, G. (2020, February). Metrics of energy consumption in software systems: a systematic literature review. In *IOP Conference Series: Earth and Environmental Science* (Vol. 431, No. 1, p. 012051). IOP Publishing.
14. Fonseca, A., Kazman, R., & Lago, P. (2019). A manifesto for energy-aware software. *IEEE Software*, *36*(6), 79-82.
15. Fontana, F. A., Ferme, V., Zanoni, M., & Roveda, R. (2015, October). Towards a prioritization of code debt: A code smell intensity index. In *2015 IEEE 7th International Workshop on Managing Technical Debt (MTD)* (pp. 16-24). IEEE.
16. Fowler, M. (2018). *Refactoring: improving the design of existing code*. Addison-Wesley Professional.
17. Guamán, D., Pérez, J., & Correa, R. (2020). Herramienta para la personalización y cálculo de métricas de código utilizando análisis estático: SCAT. Revista Ibérica de Sistemas e Tecnologias de Informação, (E28), 693-710.
18. Guamán, D., Pérez, J., Garbajosa, J., & Rodríguez, G. (2020, November). A Systematic-Oriented Process for Tool Selection: The Case of Green and Technical Debt Tools in Architecture Reconstruction. In International Conference on Product-Focused Software Process Improvement (pp. 237-253). Springer, Cham.
19. Harnessin Green, I. T. (2012). Principles and practices. *San Murugesan*.

20. JDepend https://github.com/clarkware/jdepend, (April 2021)
21. Joulemeter, https://www.microsoft.com/en-us/research/project/joulemeter-computational-energy-measurement-and-optimization/ (April, 2020)
22. Kruchten, P., Nord, R. L., & Ozkaya, I. (2012). Technical debt: From metaphor to theory and practice. *Ieee software*, *29*(6), 18-21.
23. Lago, P., Gu, Q., & Bozzelli, P. (2014). A systematic literature review of green software metrics.
24. Li, H. (2012, April). Dynamic analysis of object-oriented software complexity. Int. Conf. on Consumer Electronics, Communications and Networks, CECNet, pp. 1791–1794, 2012.
25. Li, Z., Liang, P., & Avgeriou, P. (2015, May). Architectural technical debt identification based on architecture decisions and change scenarios. In *2015 12th Working IEEE/IFIP Conference on Software Architecture* (pp. 65-74). IEEE.
26. OMG, "Meta object facility (MOF)," 2016. http://www.omg.org/spec/MOF/2.5.1/.
27. Paige, R. F., Drivalos, et al. (2011). Rigorous identification and encoding of trace-links in model-driven engineering. *Software & Systems Modeling*, *10*(4), 469-487.
28. Pérez-Castillo, R., & Piattini, M. (2014). Analyzing the harmful effect of god class refactoring on power consumption. *IEEE software*, *31*(3), 48-54.
29. Rocheteau, J. (2015). Energy Wasting Rate as a Metrics for Green Computing and Static Analysis.
30. Singh, S., & Kahlon, K. S. (2011). Effectiveness of encapsulation and object-oriented metrics to refactor code and identify error prone classes using bad smells. *ACM SIGSOFT Software Engineering Notes*, *36*(5), 1-10.
31. SonarQube https://www.sonarqube.org/, (April 2021)
32. Srivastav, V. S. P., & Prakash, P. (2013, December). Green metrics for OO codes: CAAEC metric. In *2013 International Conference on Green Computing, Communication and Conservation of Energy (ICGCE)* (pp. 296-298). IEEE.
33. STAN, http://stan4j.com/, (April 2021)
34. Stevanetic, S., Plakidas, K., Ionescu, T. B., Schall, D., & Zdun, U. (2016, November). Supporting quality-driven architectural design decisions in software ecosystems. In *Proccedings of the 10th European Conf. on Software Architecture Workshops* (pp. 1-4).
35. Systems and software engineering — Architecture description ISO/IEC/IEEE 42010, 2011. [Online]. Available: http://www.iso-architecture.org/42010/index.html.
36. Tang, A., Avgeriou, P., Jansen, A., Capilla, R., & Babar, M. A. (2010). A comparative study of architecture knowledge management tools. *Journal of Systems and Software*, *83*(3), 352-370.
37. Tang, A., Jin, Y., & Han, J. (2007). A rationale-based architecture model for design traceability and reasoning. *Journal of Systems and Software*, *80*(6), 918-934.
38. Thamburaj, T. F., & Aloysius, A. (2017, February). Models for Maintenance Effort Prediction with Object-Oriented Cognitive Complexity Metrics. In *2017 World Congress on Computing and Communication Technologies (WCCCT)* (pp. 191-194). IEEE.
39. Tom, E., Aurum, A., & Vidgen, R. (2013). An exploration of technical debt. *Journal of Systems and Software*, *86*(6), 1498-1516.
40. Venters, C. C., Capilla, R., Betz, S., Penzenstadler, B., Crick, T., Crouch, S. & Carrillo, C. (2018). Software sustainability: Research and practice from a software architecture viewpoint. *Journal of Systems and Software*, *138*, 174-188.
41. Venters, C., et al. (2014). The blind men and the elephant: Towards an empirical evaluation framework for software sustainability. *Journal of Open Research Software*, *2*(1).
42. Vetro, A., Ardito, L., & Morisio, M. (2013). Definition, implementation and validation of energy code smells: an exploratory study on an embedded system.
43. Villa, L., Cabezas, I., Lopez, M., & Casas, O. (2016, July). Towards a sustainable architectural design by an adaptation of the architectural driven design method. In *Int. Conf. on Computational Science and Its Applications* (pp. 71-86). Springer, Cham.
44. Welter, M., Benitti, F. B. V., & Thiry, M. (2014, September). Green metrics to software development organizations: A systematic mapping. In *2014 XL Latin American Computing Conference (CLEI)* (pp. 1-7). IEEE.